Apache HTTP Server Version 1.3

Dynamic Content with CGI

- Dynamic Content with CGI
- Configuring Apache to permit CGI
  - ScriptAlias
  - CGI outside of ScriptAlias directories
    - Explicitly using Options to permit CGI execution
    - .htaccess files
- Writing a CGI program
  - Your first CGI program
- But it's still not working!
  - File permissions
  - Path information
  - Syntax errors
  - Error logs
- What's going on behind the scenes?
  - Environment variables
  - STDIN and STDOUT
- CGI modules/libraries
- For more information

Dynamic Content with CGI

Related Modules	Related Directives
mod_alias	AddHandler
mod_cgi	Options
	ScriptAlias

The CGI (Common Gateway Interface) defines a way for a web server to interact with external content-generating programs, which are often referred to as CGI programs or CGI scripts. It is the simplest, and most common, way to put dynamic content on your web site. This document will be an introduction to setting up CGI on your Apache web server, and getting started writing CGI programs.

Configuring Apache to permit CGI

In order to get your CGI programs to work properly, you'll need to have Apache configured to permit

CGI execution. There are several ways to do this.

ScriptAlias

The `ScriptAlias` directive tells Apache that a particular directory is set aside for CGI programs. Apache will assume that every file in this directory is a CGI program, and will attempt to execute it, when that particular resource is requested by a client.

The `ScriptAlias` directive looks like:

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

The example shown is from your default `httpd.conf` configuration file, if you installed Apache in the default location. The `ScriptAlias` directive is much like the `Alias` directive, which defines a URL prefix that is to mapped to a particular directory. `Alias` and `ScriptAlias` are usually used for directories that are outside of the `DocumentRoot` directory. The difference between `Alias` and `ScriptAlias` is that `ScriptAlias` has the added meaning that everything under that URL prefix will be considered a CGI program. So, the example above tells Apache that any request for a resource beginning with `/cgi-bin/` should be served from the directory `/usr/local/apache/cgi-bin/`, and should be treated as a CGI program.

For example, if the URL `http://www.example.com/cgi-bin/test.pl` is requested, Apache will attempt to execute the file `/usr/local/apache/cgi-bin/test.pl` and return the output. Of course, the file will have to exist, and be executable, and return output in a particular way, or Apache will return an error message.

CGI outside of ScriptAlias directories

CGI programs are often restricted to `ScriptAlias`'ed directories for security reasons. In this way, administrators can tightly control who is allowed to use CGI programs. However, if the proper security precautions are taken, there is no reason why CGI programs cannot be run from arbitrary directories. For example, you may wish to let users have web content in their home directories with the `UserDir` directive. If they want to have their own CGI programs, but don't have access to the main `cgi-bin` directory, they will need to be able to run CGI programs elsewhere.

Explicitly using Options to permit CGI execution

You could explicitly use the `Options` directive, inside your main server configuration file, to specify that CGI execution was permitted in a particular directory:

```
<Directory /usr/local/apache/htdocs/somedir>
        Options +ExecCGI
</Directory>
```

The above directive tells Apache to permit the execution of CGI files. You will also need to tell the server what files are CGI files. The following `AddHandler` directive tells the server to treat all files with the `cgi` or `pl` extension as CGI programs:

```
AddHandler cgi-script cgi pl
```

.htaccess files

A `.htaccess` file is a way to set configuration directives on a per-directory basis. When Apache serves a resource, it looks in the directory from which it is serving a file for a file called `.htaccess`, and, if it finds it, it will apply directives found therein. `.htaccess` files can be permitted with the `AllowOverride` directive, which specifies what types of directives can appear in these files, or if they are not allowed at all. To permit the directive we will need for this purpose, the following configuration will be needed in your main server configuration:

```
AllowOverride Options
```

In the `.htaccess` file, you'll need the following directive:

```
Options +ExecCGI
```

which tells Apache that execution of CGI programs is permitted in this directory.

Writing a CGI program

There are two main differences between ``regular'' programming, and CGI programming.

First, all output from your CGI program must be preceded by a MIME-type header. This is HTTP header that tells the client what sort of content it is receiving. Most of the time, this will look like:

```
Content-type: text/html
```

Secondly, your output needs to be in HTML, or some other format that a browser will be able to display. Most of the time, this will be HTML, but occasionally you might write a CGI program that outputs a gif image, or other non-HTML content.

Apart from those two things, writing a CGI program will look a lot like any other program that you might write.

Your first CGI program

The following is an example CGI program that prints one line to your browser. Type in the following, save it to a file called `first.pl`, and put it in your `cgi-bin` directory.

```
#!/usr/bin/perl
print "Content-type: text/html\r\n\r\n";
print "Hello, World.";
```

Even if you are not familiar with Perl, you should be able to see what is happening here. The first line tells Apache (or whatever shell you happen to be running under) that this program can be executed by feeding the file to the interpreter found at the location `/usr/bin/perl`. The second line prints the content-type declaration we talked about, followed by two carriage-return newline pairs. This puts a blank line after the header, to indicate the end of the HTTP headers, and the beginning of the body. The third line prints the string ``Hello, World.'' And that's the end of it.

If you open your favorite browser and tell it to get the address

```
http://www.example.com/cgi-bin/first.pl
```

or wherever you put your file, you will see the one line `Hello, World.` appear in your browser window. It's not very exciting, but once you get that working, you'll have a good chance of getting just about anything working.

But it's still not working!

There are four basic things that you may see in your browser when you try to access your CGI program from the web:

The output of your CGI program
: Great! That means everything worked fine.

The source code of your CGI program or a "POST Method Not Allowed" message
: That means that you have not properly configured Apache to process your CGI program. Reread the section on configuring Apache and try to find what you missed.

A message starting with "Forbidden"
: That means that there is a permissions problem. Check the Apache error log and the section below on file permissions.

A message saying "Internal Server Error"
: If you check the Apache error log, you will probably find that it says "Premature end of script headers", possibly along with an error message generated by your CGI program. In this case, you will want to check each of the below sections to see what might be preventing your CGI program from emitting the proper HTTP headers.

File permissions

Remember that the server does not run as you. That is, when the server starts up, it is running with the permissions of an unprivileged user - usually ``nobody", or ``www" - and so it will need extra permissions to execute files that are owned by you. Usually, the way to give a file sufficient permissions to be executed by ``nobody" is to give everyone execute permission on the file:

```
chmod a+x first.pl
```

Also, if your program reads from, or writes to, any other files, those files will need to have the correct permissions to permit this.

The exception to this is when the server is configured to use suexec. This program allows CGI programs to be run under different user permissions, depending on which virtual host or user home directory they are located in. Suexec has very strict permission checking, and any failure in that checking will result in your CGI programs failing with an "Internal Server Error". In this case, you will need to check the suexec log file to see what specific security check is failing.

Path information

When you run a program from your command line, you have certain information that is passed to the shell without you thinking about it. For example, you have a path, which tells the shell where it can look for files that you reference.

When a program runs through the web server as a CGI program, it does not have that path. Any programs that you invoke in your CGI program (like 'sendmail', for example) will need to be specified by a full path, so that the shell can find them when it attempts to execute your CGI program.

A common manifestation of this is the path to the script interpreter (often `perl`) indicated in the first line of your CGI program, which will look something like:

```
#!/usr/bin/perl
```

Make sure that this is in fact the path to the interpreter.

Syntax errors

Most of the time when a CGI program fails, it's because of a problem with the program itself. This is particularly true once you get the hang of this CGI stuff, and no longer make the above two mistakes. Always attempt to run your program from the command line before you test if via a browser. This will eliminate most of your problems.

Error logs

The error logs are your friend. Anything that goes wrong generates message in the error log. You should always look there first. If the place where you are hosting your web site does not permit you access to the error log, you should probably host your site somewhere else. Learn to read the error logs, and you'll find that almost all of your problems are quickly identified, and quickly solved.

What's going on behind the scenes?

As you become more advanced in CGI programming, it will become useful to understand more about what's happening behind the scenes. Specifically, how the browser and server communicate with one another. Because although it's all very well to write a program that prints ``Hello, World.", it's not particularly useful.

Environment variables

Environment variables are values that float around you as you use your computer. They are useful things like your path (where the computer searches for a the actual file implementing a command when you type it), your username, your terminal type, and so on. For a full list of your normal, every day environment variables, type `env` at a command prompt.

During the CGI transaction, the server and the browser also set environment variables, so that they can communicate with one another. These are things like the browser type (Netscape, IE, Lynx), the server

type (Apache, IIS, WebSite), the name of the CGI program that is being run, and so on.

These variables are available to the CGI programmer, and are half of the story of the client-server communication. The complete list of required variables is at http://hoohoo.ncsa.uiuc.edu/cgi/env.html

This simple Perl CGI program will display all of the environment variables that are being passed around. Two similar programs are included in the `cgi-bin` directory of the Apache distribution. Note that some variables are required, while others are optional, so you may see some variables listed that were not in the official list. In addition, Apache provides many different ways for you to add your own environment variables to the basic ones provided by default.

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
foreach $key (keys %ENV) {
     print "$key --> $ENV{$key}<br>";
}
```

STDIN and STDOUT

Other communication between the server and the client happens over standard input (STDIN) and standard output (STDOUT). In normal everyday context, STDIN means the keyboard, or a file that a program is given to act on, and STDOUT usually means the console or screen.

When you POST a web form to a CGI program, the data in that form is bundled up into a special format and gets delivered to your CGI program over STDIN. The program then can process that data as though it was coming in from the keyboard, or from a file

The ``special format'' is very simple. A field name and its value are joined together with an equals (=) sign, and pairs of values are joined together with an ampersand (&). Inconvenient characters like spaces, ampersands, and equals signs, are converted into their hex equivalent so that they don't gum up the works. The whole data string might look something like:

```
name=Rich%20Bowen&city=Lexington&state=KY&sidekick=Squirrel%20Monkey
```

You'll sometimes also see this type of string appended to the a URL. When that is done, the server puts that string into the environment variable called QUERY_STRING. That's called a GET request. Your HTML form specifies whether a GET or a POST is used to deliver the data, by setting the METHOD attribute in the FORM tag.

Your program is then responsible for splitting that string up into useful information. Fortunately, there are libraries and modules available to help you process this data, as well as handle other of the aspects of your CGI program.

CGI modules/libraries

When you write CGI programs, you should consider using a code library, or module, to do most of the grunt work for you. This leads to fewer errors, and faster development.

If you're writing CGI programs in Perl, modules are available on CPAN. The most popular module for this purpose is CGI.pm. You might also consider CGI::Lite, which implements a minimal set of functionality, which is all you need in most programs.

If you're writing CGI programs in C, there are a variety of options. One of these is the CGIC library, from http://www.boutell.com/cgic/

For more information

There are a large number of CGI resources on the web. You can discuss CGI problems with other users on the Usenet group comp.infosystems.www.authoring.cgi. And the -servers mailing list from the HTML Writers Guild is a great source of answers to your questions. You can find out more at http://www.hwg.org/lists/hwg-servers/

And, of course, you should probably read the CGI specification, which has all the details on the operation of CGI programs. You can find the original version at the NCSA and there is an updated draft at the Common Gateway Interface RFC project.

When you post a question about a CGI problem that you're having, whether to a mailing list, or to a newsgroup, make sure you provide enough information about what happened, what you expected to happen, and how what actually happened was different, what server you're running, what language your CGI program was in, and, if possible, the offending code. This will make finding your problem much simpler.

Note that questions about CGI problems should **never** be posted to the Apache bug database unless you are sure you have found a problem in the Apache source code.
